Efficient Learning of Bounded-Treewidth Bayesian Networks from Complete and Incomplete Data Sets

Mauro Scanagatta

IDSIA, Switzerland

Giorgio Corani

IDSIA, Switzerland

Marco Zaffalon

IDSIA, Switzerland

Jaemin Yoo

Seoul National University

U Kang

Seoul National University

Abstract

Learning a Bayesian networks with bounded treewidth is important for reducing the complexity of the inferences. We present a novel anytime algorithm (k-MAX) method for this task, which scales up to thousands of variables. Through extensive experiments we show that it consistently yields higher-scoring structures than its competitors on complete data sets. We then consider the problem of structure learning from incomplete data sets. This can be addressed by structural EM, which however is computationally very demanding. We thus adopt the novel k-MAX algorithm in the maximization step of structural EM, obtaining an efficient computation of the expected sufficient statistics. We test the resulting structural EM method on the task of imputing missing data, comparing it against the state-of-the-art approach based on random forests. Our approach achieves the same imputation accuracy of the competitors, but in about one tenth of the time. Furthermore we show that it has worst-case complexity linear in the input size, and that it is easily parallelizable.

Keywords: Structural learning, bounded treewidth, Bayesian networks, structural EM, incomplete data sets.

Email addresses: `mauro@idsia.ch` (Mauro Scanagatta), `giorgio@idsia.ch` (Giorgio Corani), `zaffalon@idsia.ch` (Marco Zaffalon), `jaeminyoo@snu.ac.kr` (Jaemin Yoo), `ukang@snu.ac.kr` (U Kang)

1. Introduction

The size of an explicit representation of the joint distribution of n categorical random variables is exponential in n. Bayesian networks [1] compactly represent joint distributions by exploiting independence relations and encoding them into a directed acyclic graph (DAG), also referred to as *structure*. Yet, algorithms able to perform structure learning from thousands of variables have been devised only very recently for Bayesian networks [2, 3] and for chordal log-linear graphical models (that can be exactly mapped on Bayesian networks) [4, 5].

Given a Bayesian network, the task of computing the marginal distribution of a set of variables, possibly given evidence on another set of variables, is called *inference*. The complexity of exact inference grows exponentially in the *treewidth* [1, Chap. 7] of the DAG, under the exponential time hypothesis [6]. In order to allow tractable inference we thus need to learn Bayesian networks with a bounded-treewidth structure; this problem is NP-hard [7].

Most research on learning bounded-treewidth Bayesian networks adopts a score-based approach. The score measures the fit of the DAG to the data; the goal is hence to find the highest-scoring DAG that respects the treewidth bound. Exact methods [7, 8, 9] exist, but their applicability is restricted to small domains. Approximate approaches that scale up to some hundreds of variables [10, 11] have been more recently proposed. A recent breakthrough has been achieved by the k-greedy algorithm [3]. It consistently yields higher-scoring DAGs than its competitors and it scales to several *thousands* of variables.

In this paper we present a new algorithm called k-MAX, which improves over k-greedy. Both k-MAX and k-greedy are anytime algorithms: they can be stopped at any moment, yielding the current best solution. k-MAX adopts a set of more sophisticated heuristics compared to k-greedy; as a result it consistently yields higher-scoring DAGs than both k-greedy and other competitors, as demonstrated by our extensive experiments on complete data sets.

Structure learning algorithms commonly assume data sets to be complete; yet real data sets are often incomplete. Structure learning on incomplete data sets can be accomplished via the *structural expectation-maximization* (SEM) algorithm [12], which alternates between an estimation of the sufficient statistics given the current model (expectation step), and the search of a new model given the expected sufficient statistics (maximization step). Yet, SEM is computationally demanding: in particular the expectation step requires computing several inferences, which might become prohibitive if the model has unbounded treewidth and/or there are many missing data whose actual value has to be inferred. We adopt k-MAX as the structure learning algorithm within SEM; in this way we obtain a fast implementation of SEM, since the bounded-treewidth structures learned in the different iterations perform efficient inferences. To the best of our knowledge, this is the first implementation of SEM that is able to scale to thousands of variables.

2

To test our method, we use the Bayesian networks learned by SEM in order to perform data imputation. We consider as a competitor a recent method for data imputation based on random forests [13] and we compare the two approaches on data sets with different degrees of missingness. The two approaches achieve the same imputation accuracy, but our approach is faster by almost one order of magnitude. Furthermore we show that the complexity of our method scales linearly in the input size (Subsec. 7.4), and that it is easily parallelizable (Subsec. 7.5). To the best of our knowledge, it is the first approach in the literature able to do so.

In Section 2 we present the technical background of the paper. In Section 3 we detail our approach for bounded-treewidth structure learning, k-MAX. In Section 4 and 5 we evaluate its performance against existing state-of-the-art approaches. In Section 6 we present how k-MAX can be used in the SEM algorithm, obtaining the SEM-k-MAX algorithm. It is evaluated in Section 7 on the task of data imputation against the state-of-the-art approach. Section 8 concludes our paper.

The software of this paper is available from `http://ipg.idsia.ch/software/blip`, together with supplementary material containing the detailed results of our experiments.

2. Treewidth and k-trees

Intuitively, the treewidth k quantifies the extent to which a graph resembles a tree. Following the terminology of [14] we now provide a formal definition. Let us recall that a *clique* of an undirected graph is a subset of its nodes such that every two distinct nodes are linked by an edge. Moreover, a clique is *maximal* if it is not a subset of a larger clique.

Treewidth of an undirected graph. We denote an undirected graph by $H = (V, E)$ where V is the vertex set and E is the edge set. An undirected graph is *triangulated* when every cycle of length greater than or equal to 4 has a *chord*, that is, an edge connecting two non-consecutive nodes in the cycle [1, Def. 9.16]. Triangulated graphs are also called *chordal* graphs. The *triangulation* of a graph is the operation of adding chords until the graph is triangulated. The treewidth of a triangulated graph is the size of its largest clique minus one. The treewidth of H is the minimum treewidth among all the possible triangulations of H.

Treewidth of a Bayesian network. The moral graph of the DAG associated to a Bayesian network is an undirected graph that includes an edge $(i - j)$ for every edge $(i \rightarrow j)$ in the DAG and an edge $(p - q)$ for every pair of edges $(p \rightarrow i)$, $(q \rightarrow i)$ in the DAG. The treewidth of the DAG is the treewidth of its moral graph.

2.1. k-trees

A k-tree is an undirected *edge-maximal* graph of treewidth k, that is, the addition of any edge to the k-tree increases its treewidth. It is defined inductively as follows [15]. *Base case*: a clique with $(k + 1)$ nodes is a k-tree. *Inductive*

3

step: given a k-tree H_n on n nodes, a k-tree H_{n+1} on $(n+1)$ nodes is obtained by connecting the $(n+1)$-th node to a k-clique of H_n (a k-clique is a clique over k nodes). See Figure 1 for an example. As a final remark, a sub-graph of a k-tree is called *partial k-tree*; its treewidth is at most k.
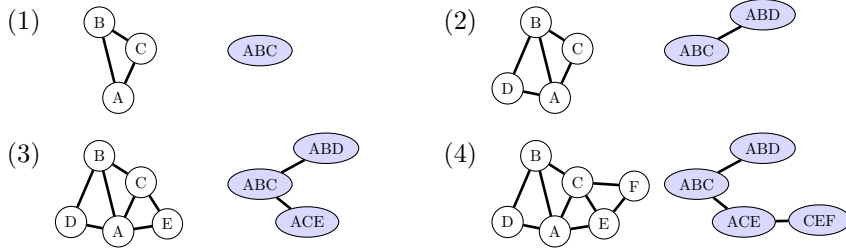

Figure 1: Iterative construction of a k-tree (k = 2). We start with the clique over the nodes {A,B,C}; then we add the nodes D, E and F one at a time. When we add a node, we link it to a 2-clique of the existing graph. On the right we show the decomposition of the graph into its maximal cliques. All maximal cliques have size three; thus the treewidth of the graph is two.

3. Structure learning of Bayesian networks

We consider the problem of learning the structure of a Bayesian network from a complete data set. The set of n categorical random variables is $\mathcal{X} = \{X_1, ..., X_n\}$. The goal is to find the highest-scoring bounded-treewidth DAG $\mathcal{G} = (V, E)$, where V is the collection of nodes and E is the collection of arcs. E can be represented by the set of parents $\Pi_1, ..., \Pi_n$ of all variables.

Structure learning is usually accomplished in two steps. First, *parent set identification* is the identification of a list (*cache*) L_i of candidate parent sets independently for each variable X_i. Second, *structure optimization* is the assignment of a parent set to each node in order to maximize the score of the resulting DAG.

The problem of bounded-treewidth structure learning can be casted as follows:

$$\mathcal{G}^* = \arg\max score(\mathcal{G}) \text{ s.t.}$$
$$\forall_i : \Pi_i \in L_i$$
$$\mathcal{G} \text{ is DAG}$$
$$treewidth(\mathcal{G}) \leq k.$$

Different scores can be used to assess the fit of a DAG. We adopt the *Bayesian information criterion* (BIC), which is asymptotically proportional to the posterior probability of the DAG. The BIC score is defined as follows:

$$\text{BIC}(\mathcal{G}) = \sum_{i=1}^{n} \text{BIC}(X_i, \Pi_i) = \sum_{i=1}^{n} \left(\text{LL}(X_i | \Pi_i) + \text{Pen}(X_i, \Pi_i) \right) , \qquad (1)$$

where $\mathrm{LL}(X_i|\Pi_i)$ denotes the log-likelihood of X_i and its parent set:

$$\mathrm{LL}(X_i|\Pi_i) = \sum_{\pi \in \Pi_i, \ x \in X_i} N_{x,\pi} \log \hat{\theta}_{x|\pi} \ , \tag{2}$$

while $\mathrm{Pen}(X_i, \Pi_i)$ is the *complexity penalization*:

$$\mathrm{Pen}(X_i, \Pi_i) = -\frac{\log N}{2}(|X_i| - 1)(|\Pi_i|) \ . \tag{3}$$

We denote by $\hat{\theta}_{x|\pi}$ the maximum likelihood estimate of the conditional probability $P(X_i = x|\Pi_i = \pi)$; by $N_{x,\pi}$ the number of times that $(X = x \wedge \Pi_i = \pi)$ appears in the data set; $|\cdot|$ indicates the size of the Cartesian product space of the variables given as argument. Thus $|X_i|$ is the number of states of X_i and $|\Pi_i|$ is the product of the number of states of the parents of X_i.

The BIC score is *decomposable*, namely it is constituted by the sum of the scores of the individual variables given their parents. The k-MAX algorithm, which we present later, can be applied to any decomposable scoring functions; see [16] for a discussion of decomposable scoring functions.

3.1. Parent set identification

In order to efficiently prepare the cache L_i of candidate parent sets of each variable X_i we adopt the approach of [2]. The main idea of [2] is to quickly identify the most promising parent sets through an approximate scoring function that does not require scanning the data set. The approximate scoring function is called BIC*. The BIC* of a parent set $\Pi = \Pi_1 \cup \Pi_2$ constituted by the union of two non-empty and disjoint parent sets Π_1 and Π_2 is:

$$\mathrm{BIC}^*(X, \Pi_1, \Pi_2) = \mathrm{BIC}(X, \Pi_1) + \mathrm{BIC}(X, \Pi_2) + \mathrm{inter}(X, \Pi_1, \Pi_2) , \tag{4}$$

that is, the sum of the BIC scores of the two parent sets and of an interaction term, which ensures that the penalty term of $\mathrm{BIC}^*(X, \Pi_1, \Pi_2)$ matches the penalty term of $\mathrm{BIC}(X, \Pi_1 \cup \Pi_2)$. In particular, If $\mathrm{BIC}(X, \Pi_1)$ and $\mathrm{BIC}(X, \Pi_2)$ are known, then BIC* is computed in constant time (with respect to data accesses). The *independence selection* algorithm [2] exploits BIC* to quickly approximately score a large number of parent sets without limiting the in-degree, which is the maximum number of parents allowed for every node. Eventually, it computes the actual score of the most promising parent sets. Additionally we adopt pruning rules [17] in order to avoid computing the score of sub-optimal parent sets.

3.2. Learning Bayesian networks with bounded treewidth

Exact methods for bounded-treewidth structure learning of Bayesian networks [7, 8, 9] scale to at most a few dozens of variables. Approximate approaches are therefore needed to scale to larger domains.

The S2 algorithm [10] uniformly samples the space of k-trees; then it assesses the sampled k-trees through a heuristic scoring function (*informative score*).

The DAG is then obtained by constraining its moral graph to be a sub-graph of the k-tree with highest informative score. The S2+ algorithm [11] further refines this idea, obtaining via A* the k-tree guaranteed to maximize the informative score. In general S2+ recovers higher-scoring DAGs than S2 but its scalability is limited: for instance it cannot be used with thousands of variables.

3.2.1. k-greedy

To the best of our knowledge, the state-of-the-art algorithm for bounded-treewidth learning is so far constituted by *k-greedy* [3], which consistently yields higher-scoring DAGs than its competitors. The k-greedy algorithm samples the space of the orderings of variables, and given an ordering it builds the bounded-treewidth DAG inductively as follows.

Initialization. Given an order over the variables and the value of k, k-greedy initializes the structure with a DAG over the first $(k+1)$ variables in the order. The DAG is learned using either the exact method of [18] or the approximate method of [19], depending on the value of $(k+1)$. The treewidth of the learned DAG is at most k (its moral graph is a sub-graph of the clique over the same $k+1$ variables, which has treewidth k).

Addition of the following nodes. Given a DAG $\mathcal{G}_q$ over q variables, k-greedy chooses the highest-scoring *feasible* parent set for the variable $(q+1)$ in the order, $X_{\prec q+1}$. A parent set is feasible if it is a k-clique (or a subset of a k-clique) of the moral graph of $\mathcal{G}_q$. Once the parent set of $X_{\prec q+1}$ is chosen, we obtain the DAG $\mathcal{G}_{q+1}$ over $q+1$ variables. At each iteration the moral graph of the DAG is a partial k-tree: thus the treewidth of the DAG is bounded by k.

Then k-greedy samples a new ordering and repeats the above procedure. When a maximum number of iterations or a maximum execution time is met it returns the highest-scoring DAG found.
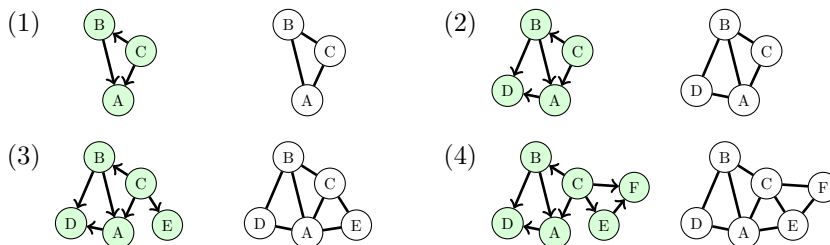
3.3. k-MAX

Figure 2: Example of a treewidth-bounded (k=2) DAG built by k-MAX. For each iteration we also show the underlying k-tree graph (white nodes), which allows to check that the treewidth is indeed bounded by k=2. At iteration (2), $m(D) > m(E)$ and $m(E) > m(F)$; hence node D is inserted. Its highest-scoring feasible parent set is constituted by {A,B}. At iteration (3) we still have $m(E) > m(F)$ and thus we insert E. Its highest-scoring parent set is constituted by only {C}, so in the k-tree we choose a k-clique that is a random superset of {C}. At iteration (4) we insert the last remaining variable F.

k-MAX shares a fundamental idea with k-greedy: it incrementally grows a DAG, guaranteeing that at each step its moral graph is a sub-graph of a k-tree. Differently from k-greedy, k-MAX does not adopt a predefined ordering over the variables. Instead it ranks the variables that can be inserted in the graph through the heuristic score $m(X_i)$:

$$m(X_i) = \frac{sc^C(X_i) - sc^W(X_i)}{sc^B(X_i) - sc^W(X_i)} \ ,$$

where:

$$sc^C(X_i) = \max_{\Pi \in L_i^*} score(\Pi) \ ,$$
$$sc^B(X_i) = \max_{\Pi \in L_i} score(\Pi) \ ,$$
$$sc^W(X_i) = \min_{\Pi \in L_i} score(\Pi) \ .$$

L_i^* is the subset of parent sets that are *feasible*. Recall that the feasible parent sets are constituted by the k-cliques of the moral graph of the current DAG, and by subsets of such k-cliques.

The $m(X_i)$ heuristic compares for each variable the highest-scoring *feasible* parent set with the lowest-scoring and the highest-scoring parent set available in the cache L_i (notice that most parent sets in L_i are not feasible when the k-tree contains a small number of variables). The rationale is to defer the addition of variables whose $m(\cdot)$ score is low, as it will increase in the subsequent iterations due to availability of a larger number of feasible parent sets. The scores $sc^B(X_i)$ and $sc^W(X_i)$ can be found in linear time w.r.t. $|L_i|$ and cached. As for $sc^C(X_i)$, it needs to be updated each time a new variable is added to the DAG.

We present an outline of k-MAX in Alg. 1. The procedure is repeated until a specific termination condition is met (for example maximum execution time). The highest-scoring DAG found is then returned.

Algorithm 1 k-MAX

1: **procedure** K-MAX($\mathcal{X}$ // Set of variables)
2:     **while** not termination condition **do**
3:         $q \leftarrow k + 1$
4:         $\mathcal{G}_q \leftarrow Initialization(\mathcal{X})$ // Learn the initial graph with $k{+}1$ variables
5:         **while** $q < |\mathcal{X}|$ **do**
6:             $\mathcal{G}_{q+1} \leftarrow Addition(\mathcal{G}_q)$ // Add the variable with the highest $m(X_i)$ score
7:             $q \leftarrow q + 1$
8:         **end while**
9:     **end while**
10:     // Return the highest-scoring $\mathcal{G}$ found
11: **end procedure**

Initialization. We start by building an initial k-tree $\mathcal{K}_{k+1}$ over $k + 1$ variables as follows. We initialize the list of chosen variables $\mathcal{I}$ as an empty set, and we create a set $\mathcal{C}$, which stores the *candidate* parents: namely every variable appearing at least in one parent set of $X_i \in \mathcal{I}$. We first choose randomly the first variable X, we add it to the set $\mathcal{I}$ and we add its candidate parents to $\mathcal{C}$. Then, until $\mathcal{I}$ contains $k + 1$ variables we: (1) add to it a new variable Z chosen randomly from $\mathcal{C}$; (2) add all the candidate parents of Z to $\mathcal{C}$.

Finally we learn the initial DAG $\mathcal{G}_{k+1}$ over the variables contained in $\mathcal{I}$, either with exact or approximate methods, as in the initialization of k-greedy. The moral graph of $\mathcal{G}_{k+1}$ is a sub-graph of $\mathcal{K}_{k+1}$ and thus $\mathcal{G}_{k+1}$ has treewidth at most k. For each new k-clique added to $\mathcal{K}_{k+1}$, we update $m(X_i)$ for each variable X not yet processed.

Addition of the following nodes. Let us denote by $\mathcal{G}_q$ and $\mathcal{K}_q$ the current DAG and the k-tree over q nodes.

The $(q + 1)$-th variable to be added is:

$$X = \arg\max_{Z \notin G} m(Z) \ .$$

We connect X to the parent set:

$$\Pi_X^C = \arg\max_{\Pi \in L_i^*} score(\Pi) \ .$$

This yields the updated DAG $\mathcal{G}_{q+1}$. We then update the k-tree, connecting X to the k-clique that is superset of Π_X^C. In the event of several k-cliques sharing this property, a random one between them is chosen. This yields the k-tree $\mathcal{K}_{q+1}$; it contains an additional $(k + 1)$-clique compared to $\mathcal{K}_q$. By construction, $\mathcal{K}_{q+1}$ is also a k-tree. For each new k-clique added to $\mathcal{K}_{q+1}$, we update $m(X_i)$ for each variable X_i not yet processed.

Space of learnable DAGs. A reverse topological order is an order $X_1, ... X_n$ over the vertices of a DAG in which each vertex X_i appears before its parents Π_i. The search space of k-MAX contains only DAGs whose reverse topological order, when used as variable elimination order, has treewidth k. The proof is identical to that provided in [3] for k-greedy. The extensive experiments by [3] show that such limitation does not hurt the empirical performance of k-greedy and thus we do not further discuss this point.

3.4. Advantages over k-greedy

Initialization. Recall that k-MAX builds iteratively the initial set of $(k + 1)$ variables by ensuring that any added variable is a candidate parent of at least another variable already in the clique.

We prune the available parent sets for each variable by Lemma 1 of [20]. It states that given two parent sets Π_i^1 and Π_i^2 for the same variable X_i, such that $\Pi_i^2 \subseteq \Pi_i^1$ and $score(\Pi_i^2) < score(\Pi_i^1)$, then Π_i^1 can be discarded from L_i as it yields sub-optimal structures. This happens when Π_i^1 has low mutual

information with X_i; its contribution to the score is negative since the small increase in log-likelihood (due to low mutual information) is outweighed by the complexity penalization of the scoring function. Thus a variable is a *candidate parent* only if it appears in at least one non-pruned parent set of X_i (and which thus has significant mutual information with X_i).

Instead the initialization of k-greedy randomly samples $k+1$ variables, ignoring their mutual information. By virtue of the selection process the initial DAG found by k-MAX is in general higher-scoring than the one found by k-greedy.

Addition of the following nodes. The addition step of k-greedy follows the order, which is randomly sampled. Again, this overlooks whether there exists a feasible parent set given the current k-tree with a good score for the variable being added. k-MAX instead optimizes the variable to be added at each iteration, by ranking them according to the $m(\cdot)$ score.

4. Experiments: k-MAX against k-greedy

Table 1: The 18 real data sets used in the experiments (n is the number of variables, d is the number of data points).

Name	n	d	Name	n	d	Name	n	d
Kdd	64	11490	Retail	135	4408	EachMovie	500	591
Plants	69	3482	Pumsb-star	163	2452	WebKB	839	838
Audio	100	3000	DNA	180	1186	Reuters-52	889	1540
Jester	100	4116	Kosarek	190	6675	C20NG	910	3764
Netflix	100	2634	MSWeb	294	5000	BBC	1058	330
Accidents	111	2551	Book	500	1739	Ad	1556	491

Table 2: The 20 known networks from which we sampled synthetic data sets (n is the number of variables).

Name	n	Name	n	Name	n
andes	223	r2	2000	r9	4000
diabetes	413	r3	2000	r10	10000
pigs	441	r4	2000	r11	10000
link	724	r5	4000	r12	10000
munin	1041	r6	4000	r13	10000
r0	2000	r7	4000	r14	10000
r1	2000	r8	4000		

We consider the 18 data sets listed in Table 1. They have been previously used in [21] and in other works referenced therein. The data sets are available for instance from `https://github.com/arranger1044/awesome-spn#dataset`. Each

data set is split in three subsets; we thus perform 18·3=54 structure learning ex-
periments. The number of instances in each dataset ranges from 226 to 180000.

Additionally, we compared k-MAX and k-greedy on 20 synthetic data sets
sampled from known networks (Table 2). Five of these are taken from the
literature[1] (*andes*, *diabetes*, *pigs*, *link*, *munin*) while the other fifteen (r0-r14)
have been generated by us, using the BNgenerator package[2]. They contain up
to 10,000 variables. From each known network we sample a training data set of
5000 instances.

We hence consider a total of 74 data sets (54 real ones and 20 synthetic
ones), on which we compare k-greedy and k-MAX. We provide both algorithms
with the same cache of parent sets for each variable, pre-computed using inde-
pendence selection [2]; we then let each algorithm run for one hour on the same
machine.

In each experiment we measure the difference between the BIC scores (ΔBIC)
of the DAG returned by k-MAX and k-greedy. There is an exact mapping be-
tween the values of ΔBIC and the Bayes factor (BF) [22, Sec. 4.3]. The BIC
score of graph $\mathcal{G}$ is an approximation of the logarithm of the marginal likelihood
of $\mathcal{G}$, namely $P(D|\mathcal{G}) = \int P(D|\mathcal{G}, \theta)p(\theta)d\theta \simeq \mathrm{BIC}(\mathcal{G})$. Given two graphs $\mathcal{G}_1$ and
$\mathcal{G}_2$, the Bayes factor (BF) is the ratio of their marginal likelihoods. The log of
the Bayes factor can be approximated by the difference of the BIC scores:

$$\log(\mathrm{BF}) = \log\left(\frac{P(D|\mathcal{G}_1)}{P(D|\mathcal{G}_2)}\right) \simeq \mathrm{BIC}(\mathcal{G}_1) - \mathrm{BIC}(\mathcal{G}_2) = \Delta\mathrm{BIC}.$$

A positive ΔBIC provides evidence in favor of $\mathcal{G}_1$ and a negative ΔBIC provides
evidence in favor of $\mathcal{G}_2$. The posterior probability of $\mathcal{G}_1$ is given by

$$P(\mathcal{G}_1|D) = \frac{P(D|\mathcal{G}_1)}{P(D|\mathcal{G}_1) + P(D|\mathcal{G}_2)} \simeq \frac{\exp(\mathrm{BIC}(\mathcal{G}_1))}{\exp(\mathrm{BIC}(\mathcal{G}_1)) + \exp(\mathrm{BIC}(\mathcal{G}_2))}$$

For instance a ΔBIC >10 implies a Bayes factor >150 and a posterior probability
$P(\mathcal{G}_1|D)$ >0.99, conveying very strong evidence in favor of $\mathcal{G}_1$. Following the
same logic, the values of ΔBIC can be interpreted [22, Sec. 4.3] according to
this scale:

- ΔBIC >10: extremely positive evidence;

- 6 <ΔBIC <10: strongly positive evidence;

- 2 <ΔBIC <6: positive evidence;

- ΔBIC <2: neutral evidence.

We report only the case of positive ΔBIC; negative values of ΔBIC are inter-
preted in the same way but they have the meaning of negative evidence.

---

[1] http://www.bnlearn.com/bnrepository/
[2] http://sites.poli.usp.br/pmr/ltd/Software/BNGenerator/

Table 3: Comparison between k-greedy and k-MAX under various treewidths. For each treewidth we perform 74 structure learning experiments. The results in favor of k-MAX are statistically significant for each tested treewidth.

	treewidth		
k-MAX vs k-greedy	2	5	8
ΔBIC			
extremely positive	**71**	**67**	**69**
strongly positive	0	0	0
positive	0	0	1
neutral	0	0	0
negative	0	0	0
strongly negative	0	0	0
very negative	3	7	5

We perform independent experiments with the treewidths $k \in \{2, 5, 8\}$. We summarize the results in Table 3. In most cases there is an extremely positive evidence for the model learned by k-MAX over the model learned by k-greedy (ΔBIC >10). We further analyze the results through a sign-test, considering one method as winning over the other when there is a ΔBIC of at least 2 in its favor, and treating as ties the cases in which $|\Delta$BIC$|$ <2. The number of wins obtained by k-MAX over k-greedy is significant *for every tested treewidth*.

Iteration statistics. We further compare k-MAX and k-greedy by analyzing their iterations. We consider as an example the data set *tmovie.test* (500 variables) with treewidth k=5. As shown in Table 4, k-MAX performs much less iterations (two orders of magnitude less, in this example) than k-greedy; this is due to the overhead of updating the $m(\cdot)$ values for all the variables not yet added to the structure. However this strategy pays off, as the median score of the DAG retrieved at each iteration is much higher for k-MAX. This is the advantage of using the more sophisticated heuristics of k-MAX.

Table 4: Statistics about the execution of k-greedy and k-MAX on the *tmovie.test* dataset (500 variables).

	k-MAX	k-greedy
Number of iterations	1,111	96,226
Median BIC score	-36,937	-37,489

Comparison against S2. In [3] it was shown that k-greedy consistently outperforms S2. k-MAX further increases the gap over S2. In particular k-MAX achieves ΔBIC >10 compared to S2 for every treewidth and data set considered.

Inference times. We perform some tests about inference times, using Iterative Join Graph Propagation [23] as inference engine. We focus first on the networks containing 1,000 or more variables provided in Table 2, in which case the

11

ground-truth networks are known. Using such large ground-truth networks results in slow inference even when computing *marginals*. In several cases we had
265 no convergence of the inference even after 30 minutes of computation. In these cases, even if we could manage to learn the actual DAG with a perfect learner, the model would be hardly usable due to the slowness of the inference. Conversely, the bounded-treewidth models learned by k-MAX compute marginals consistently in less than 0.1 seconds, even with treewidth 8. By bounding the
270 treewidth we thus guarantee the efficiency of the inferences.

Similar considerations hold also for the smaller ground-truth networks, such as andes, diabetes, etc. In these cases marginals can be efficiently computed using the ground-truth networks, but slowness problems appear when we compute the probability of the *joint* evidence of *five* variables. This requires (averaging
275 over data set) about 60 seconds when using the ground-truth networks and less than 5 seconds when using bounded-treewidth models with treewidth *eight* (the *slowest* bounded-treewidth model).

5. Comparison with Chordalysis

Chordalysis [24] performs structural learning for log-linear models; such
280 undirected graphical models are also known as Markov networks. In particular Chordalysis learns chordal models, which are at the intersection between Bayesian networks and Markov networks: given a chordal Markov network it is possible to obtain a Bayesian network which encodes exactly the same independences [25, Sec. 4.5.3].
285 Chordalysis starts from the empty graph, which contains no edges. It then decides which edges to add based on a series of statistical test of independence. While classical approaches to log-linear analysis hardly scale beyond ten variables, Chordalysis scales to high-dimensional data thanks to sophisticated algorithms for efficiently computing the statistics of the test and avoid-
290 ing the computation of unnecessary tests. We learn the undirected chordal graph using the variant of Chordalysis referred in [5] (code available from `https://github.com/fpetitjean`) and we subsequently obtain the equivalent Bayesian network using the algorithm of [25, Sec. 4.5.3].

While score-based structural learning aims at maximizing the predictivity
295 of the model, structural learning based on hypothesis tests aims at building an explanatory model, by controlling the rate of false positive edges among those which constitute the graph. Thus such two approaches have different goals. Yet, it does make sense to compare k-MAX and Chordalysis. These are among the very few methods able to learn PGMs from thousands of variables; moreover,
300 even if Chordalysis does not formally bound the treewidth, it generally yields quite sparse graph that are likely to have low treewidth (see the results shown later). This happens because the statistical test (corrected for multiple comparisons) does not allow adding an arc unless there is strong evidence against the null hypothesis of independence.

5.1. Results

In this comparison we avoid using the BIC score of the resulting networks as a performance indicator, as Chordalysis does not aim at maximizing it. Instead, we measure how well the models fit the distribution by computing the log-likelihood of the instances of the test set, $LL = \sum_{i=1}^{k} log(P(D_i|M))$, where $D_1, \ldots, D_k$ denote the k instances of the test set and M is the model being tested. We then compute the difference in test-set log-likelihood (ΔLL) between the model learned by k-MAX and by Chordalysis. A value greater than 0 indicates that the model learned by k-MAX yields higher likelihood (thus better fit) than the model learned by Chordalysis, and vice versa.

We first consider the datasets listed in Table 1. Recall that each data set is split into three parts; for each dataset, we use a part for learning the models and the union of the other two parts as a test set. We thus perform 3 experiments for each data set, for a total of 54 experiments. In Fig. 3 (upper plot) we show the distribution of ΔLL across the data sets for different tested treewidths. The models learned by Chordalysis provide a fit that is comparable to the models learned by k-MAX using treewidth 2. When k-MAX adopts higher treewidths, such as 5 or 8, it fits better (or even much better) the distribution than Chordalysis.

We then consider the true networks of Table 2. In Fig. 3 (lower plot) we see that the same pattern appears for the ΔLL. In this case we sample from the networks 5000 instances as the training set and 50000 instances for the test set.
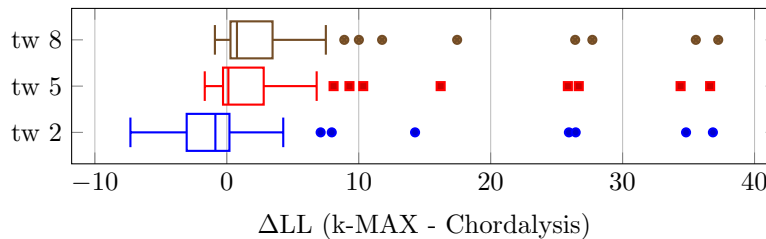
(a) Real data sets
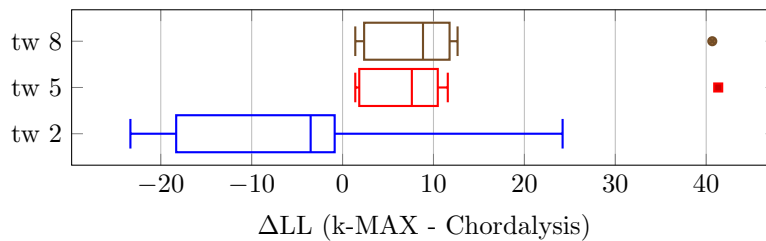
(b) Synthetic data sets

Figure 3: Difference in test-set log-likelihoods between the models learned by k-MAX (using various treewidths) and Chordalysis, across the real data sets of Table 1 and the synthetic data sets of Table 2.
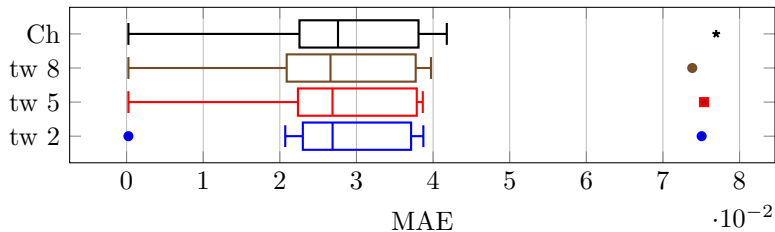
Figure 4: Comparison of the MAE of the inference (probability of the evidence of five variables) for the models yielded by Chordalysis and k-MAX under various treewidths.
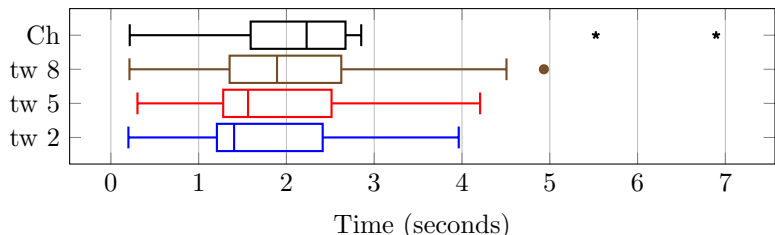

Figure 5: Time required to compute the probability of the joint evidence of five variables using the networks learned by Chordalysis and k-MAX with various treewidths.

Additionally we study the running time and the accuracy of the inference. As inference we consider the computation of the probability of the evidence $P(e)$ constituted by *five* randomly selected variables, which we set in random states. For each data set we run 100 queries. On each data set we measure the mean absolute error (MAE) of each model:

$$\text{MAE} = \frac{1}{q} \sum_i |P_i(e) - \hat{P}_i(e)| \,,$$

where q denotes the total number of queries, $P_i(e)$ and $\hat{P}_i(e)$ the probability of evidence computed by respectively the ground-truth model and the bounded-treewidth model on the i-th query. As ground-truth we take the probability of evidence computed on the original network, using the algorithm of Iterative Join Graph Propagation [23] and running it until convergence.

We show in Fig. 4 the MAE of the model learned by Chordalysis and the models learned by k-MAX with the treewidth bound in $\{2, 5, 8\}$. The MAE of the learned models are comparable, with a slight improvement with higher value of treewidth (not visible in the graph). In Fig. 5 we show the time required to compute the probability of evidence. The inference times of the models learned by k-MAX increase with the treewidth. Yet even the models learned by k-MAX with treewidth 8 are on average slightly faster than the model learned by Chordalysis. Generally the k-MAX models with treewidth 5 or 8 yield both a better fit and a quicker inference than Chordalysis. This shows the soundness of k-MAX. We recall that however Chordalysis aims at building explanatory models rather than predictive ones.

14

6. Structural EM

Most works in structure learning assume the data set to be *complete*. Yet this is rarely the case in real-word domains. This poses a serious problem, since learning a structure from *incomplete* data is computationally challenging.

The most common approach for structure learning from incomplete data sets is the *structural EM* (SEM), originally presented in [26, 27]; a discussion about this algorithm is given also by [25, Chapter 19].

The key idea of structural EM is to use the current best estimate of the distribution to complete the data, and then analyze such complete data. In a nutshell, structural EM performs search in the joint space of structure and parameters. It alternates between finding better parameters for the current structure, or select a new structure. The former case is as a parametric EM step, while the latter step is a structural EM step. For penalized likelihood scoring functions, such as the BIC, this procedure is proven to converge to a local maximum [26].

More in detail, structural EM proceeds by alternating two steps. In the *expectation* step, it computes the *expected* sufficient statistics given a candidate structure (the sufficient statistics are the counts of the occurrences of each value of a given variable jointly with each possible assignment of its parents). Given the expected sufficient statistics, the *maximization* step learns an updated structure and estimates its parameters. The two steps are alternated until the search converges to a structure. We present SEM in Alg. 2, adopting the description of [25, Chapter 19].

Algorithm 2 Structural EM algorithm

1: **procedure** STRUCTURAL-EM(
   $\mathcal{G}^0$ // Initial Bayesian network structure,
   $\boldsymbol{\theta}^0$ // Initial set of parameters for $\mathcal{G}^0$,
   $\mathcal{D}$ // Partially observed data set
)
2:     **for** each $t = 0, 1, ...,$ until convergence **do**
3:         // Optional parameter learning step
4:         $\boldsymbol{\theta}^{t'} \leftarrow$ Expectation-Maximization($\mathcal{G}^t, \boldsymbol{\theta}^t, \mathcal{D}$)
5:         // Run EM to generate expected sufficient statistics;
6:         // this yields the imputed data $\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}}$.
7:         $\mathcal{G}^{t+1} \leftarrow$ Structure-Learn($\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}}$)
8:         $\boldsymbol{\theta}^{t+1} \leftarrow$ Estimate-Parameters($\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}}, \mathcal{G}^{t+1}$)
9:     **end for**
10: **end procedure**

The most demanding part of SEM is the expectation step, which requires computing several queries, whose complexity is (in the worst case) exponential in the treewidth of the model. Thus SEM becomes prohibitive if the model being learned has unbounded treewidth and there are many missing data. This

15

prevents the application of SEM to large data sets. We adopt k-MAX in order to perform structure learning within SEM: by learning bounded-treewidth models we obtain an efficient computation of the expected sufficient statistics. We call the resulting approach as *SEM-kMAX*. As far as we know this is the first implementation of the structural expectation-maximization that scales up to thousands of variables.

The `Structure-Learn` step of Algorithm 2 is constituted by two parts. Given the (expected) sufficient statistics computed in the previous iteration, we first compute the cache of best parent sets for each variable using *independence selection*. Then we find the highest-scoring DAG using k-MAX with a treewidth bound of $k = 6$. We set an execution time of n seconds (one second per variable) for the first step and of $n/10$ seconds for the second step. Such time limits are shorter than those adopted on complete data sets, as we need to perform structure learning at each maximization step of SEM. In Section 7.3 we evaluate the sensitivity of the results on the allowed execution time of both steps.

The algorithm reaches convergence when the structure remains unchanged between the subsequent SEM iterations and hence no improvements are found on the structure of the previous iteration. As for the choice of the initial network we adopt a random chain that connects all the variables, as in [12].

6.1. Further implementation details

A peculiar aspect of our implementation is that we adopt the *hard EM* [25, Chap. 19.2.2.6] for the computation of the expected sufficient statistics. While the standard *soft* EM produces a probability distribution over the missing data, hard EM fills-in the missing data with their most probable completion. The relative merits of hard and soft EM are discussed for instance by [28] and [25, Chap. 19.2.2.6]. We adopt the hard EM in order to limit the memory usage of the algorithm. In fact we cannot foresee which sufficient statistics will be required by the maximization step, when it looks for a new structure. Keeping the soft-completed data set in memory is however not feasible: the memory requirement is, for each instance, exponential in the number of missing values. Some workarounds exist [25, Chap. 19.4.3.4], based on severe restrictions on the type of learnable structures. By adopting the hard EM we radically solve the memory issues, as the hard-completed data set requires the same memory of the original data set. This allows us to perform structure learning without restricting the search space of the structures, apart from the bounded-treewidth constraint.

Given an instance of the dataset containing missing values, we can fill the missing values *jointly* or *independently*. The joint approach requires running a single MPE (most probable explanation) query [25, Chap. 2.1.5.2] for each incomplete instance. The independent approach requires running a marginal query for each missing value, marginalizing out over the missing variables in the same instance. In the field of multi-label classification (where one has to predict a set of related labels, given the observed features) it has been pointed out [29] that the MPE inference maximizes the probability of correctly predicting

16

the *whole* set of labels, while the marginal inference maximizes the probability of correctly predicting each label independently. In practical terms, extensive experiments in multi-label classification do not show major differences between the results yielded by the two approaches [30]. There are however applications, such as message decoding over a noisy channel, where the joint approach is clearly preferable. In the following, we report results obtained using the joint approach.

7. Application to data imputation

We benchmark the performance of SEM-kMAX in the task of *data imputation*, which is the process of replacing missing data with the predictions of their values. Once we run SEM until convergence, we have both a trained model and an *imputed* data set.

A pioneering Bayesian network approach for data imputation is that of [31], which however requires to order the variables according to their reliability before performing structure learning; this approach is hardly applicable in data mining applications, where the number of variables can be in order of the thousands. As a state-of-the-art competitor for data imputation we thus consider an approach based on random forests.

The *missForest* algorithm [13] recasts the problem of missing data imputation as a prediction problem. The initial guess is made using mode imputation, namely by substituting missing values with the most common value of the variable. The variables are then sorted according to the amount of missing values and the data are imputed by regressing (using random forest) each variable in turn against all other variables (starting from the variable with the smallest missingness). The predictions of missing data for the dependent variable are used as imputation. The empirical study of [32] compares different imputation algorithms based on random forests, concluding that *missForest* is indeed the most accurate, but also the slowest. The problem is that at each iteration it requires fitting n random forests (one for each variable); this becomes slow when dealing with large number of variables. Thus [32] propose the mRF$_\alpha$ algorithm as a scalable alternative to *missForest*. It randomly divides the n variables into mutually exclusive groups of approximate size αp. Each group in turn acts as the multivariate response to be regressed on the remaining $(1 - \alpha)p$ variables at each iteration. Thus at each iteration it trains $1/\alpha$ random forest models. We used the implementation of mRF$_\alpha$ available in the *randomForestSRC* R package, setting $\alpha = 0.25$, which yields good results in the experiments of [32].

7.1. Experimental setup

We ran imputation experiments on the 18 data sets of Table 1. On each dataset we induced missing completely at random (MCAR) missingness: namely, each observation was made missing with fixed probability, regardless of the values of the other variables. For each data set we considered the missingness percentages $\{1, 2, 3, 5, 8, 10, 12, 15\}$ (meaning, e.g., that in the first setting we

made missing each value with probability 1%), and we repeated each experiment 5 times. In each repetition we measured the proportion of missing values that were correctly imputed (*imputation accuracy*):

$$\text{imputation accuracy} = \frac{1}{n} \sum_{j=1}^{n} \frac{\sum_{j=1}^{m_j} \mathbb{1}(X_{i,j}^{orig} = X_{i,j}^{imp})}{m_j} \ ,$$

where $X_{i,j}^{orig}$ denotes the value of the variable X_i in instance j in the original dataset, $X_{i,j}^{imp}$ its imputed value, m_j the number of values missing in instance j and n is the total number of instances.

7.2. Comparison with mRF$_\alpha$

In the comparison with mRF$_\alpha$ we used all the available datasets of Table 1. In two cases (*MSWeb* and *C20NG*) mRF$_\alpha$ failed to provide a solution in less than 24 hours, and the datasets were removed from the comparison. We conjecture that this is due to the presence of both a high number of variables and a high number of data points.

Figure 6 shows the scatter plots of imputation accuracy and execution times, which compare SEM-kMAX and mRF$_\alpha$ for different missingness levels. The two approaches offer practically the same imputation accuracy (left plots), but SEM-kMAX is substantially faster than mRF$_\alpha$ (right plots).

We further report their ratio of imputation accuracy in Fig. 7 and their ratio of execution times in Fig. 8 by aggregating the experiments performed on various data sets but having the same level of missingness. Again we can see that SEM-kMAX achieves an imputation accuracy comparable to mRF$_\alpha$, while on average reducing the computational time of one order of magnitude.

7.3. Parameter tuning

As mentioned, one can tune SEM-kMAX by choosing the allowed maximum execution time for its steps. We will now experimentally evaluate different possibilities as for this choice. We allow for the parent set indentification step $n{\cdot}t$ seconds, and for the structure optimization step $(n{\cdot}t)/10$ seconds, comparing the choices $t = 1$, $t = 5$ and $t = 10$. We denote respectively by SEM_1, SEM_5 and SEM_{10} the resulting variants of structural EM (in the previous experiments we adopted SEM_1, which is a good compromise between imputation accuracy and required time).

In Figure 9 we plot the total execution time for all the datasets, for the three configurations. For SEM_1 the required time is almost linear in the size of the dataset, showing that learning from incomplete data is feasible even for big datasets with thousands of variables. For SEM_5 and SEM_{10} we see that the for the largest dataset the total execution time is closer to SEM_1. The reason for this is the cost for executing the expectation step, which remains almost the same in all the cases.
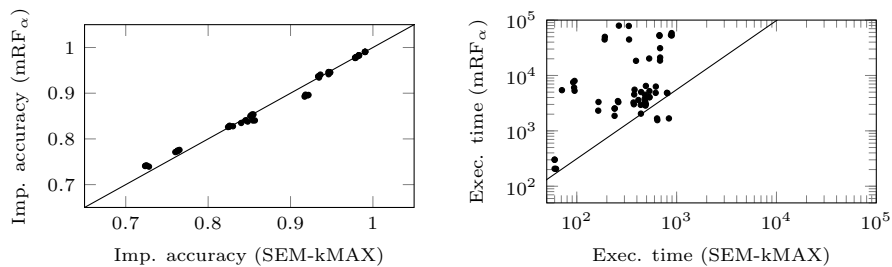
From the overall behaviour we can see that the complexity increase in the required execution time is at most linear in the number of variables. In Figure 10a
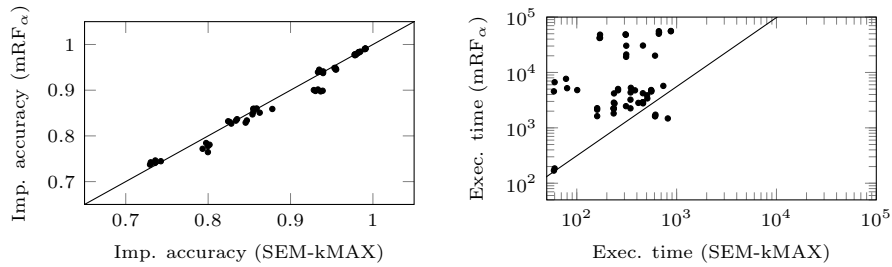
(a) Result obtained with 2% missingness (each point refers to a data set).
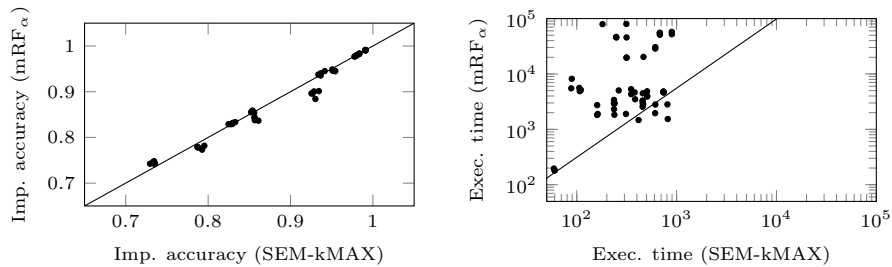
(b) Result obtained with 5% missingness (each point refers to a data set).

(c) Result obtained with 12% missingness (each point refers to a data set).

Figure 6: Comparison on imputation accuracy (on the left) and on execution times (on the right, in seconds) between imputation executed with mRF$_\alpha$ and SEM-kMAX.

we plot the imputation accuracy of SEM_5 against SEM_1 and in Figure 10b between SEM_{10} against SEM_1. The advantage of SEM_5 over SEM_1 is clear, while the advantage of SEM_{10} over SEM_5 can be called into question.

By choosing a value for the parameter t, the user can tune the trade-off between imputation accuracy and total execution time. We recommend using SEM_1 as it strikes an optimal balance between the two objectives, at least in our experiments.

7.4. Computational complexity

We now study the complexity of SEM-kMAX. Let us focus on a single iteration of the procedure. Each iteration is composed of three different steps:
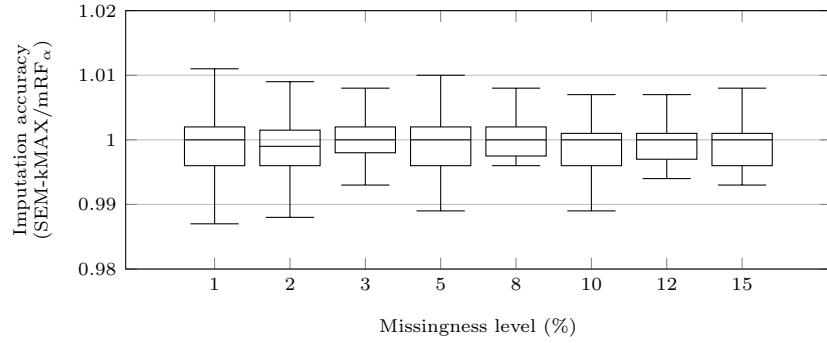
19

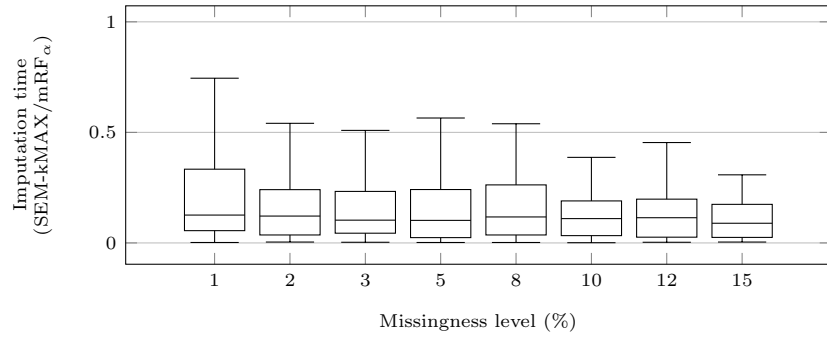Figure 7: Aggregate comparison of the imputation accuracy ratio between SEM-kMAX and mRF$_\alpha$ imputation.

Figure 8: Aggregate comparison of the execution time ratio between SEM-kMAX and mRF$_\alpha$ imputation.
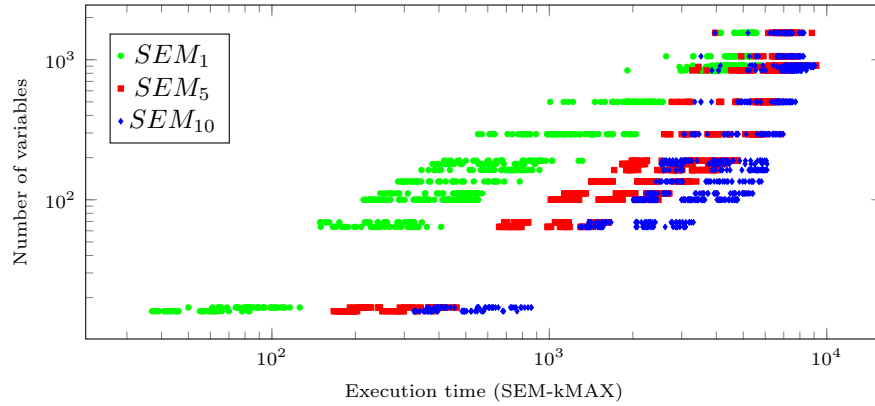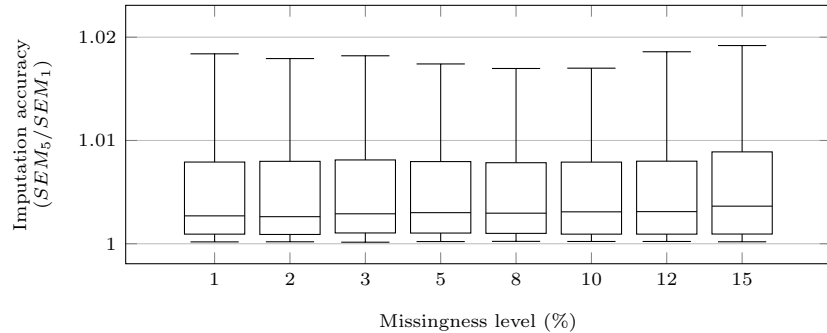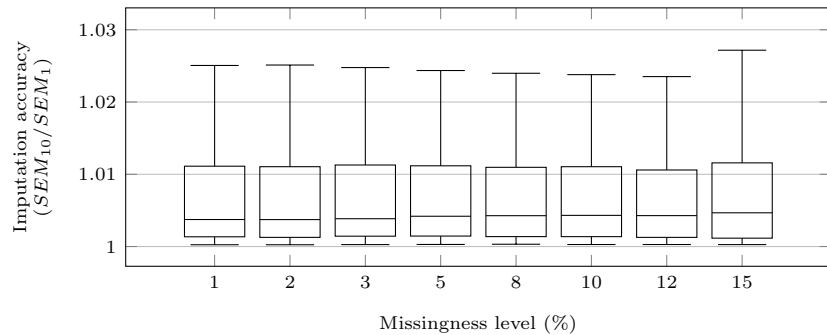

Figure 9: Execution time of SEM-kMAX in relation to the number of variables in the dataset.

(a) Aggregate comparison of the ratio of imputation accuracy between SEM_5 and SEM_1.

(b) Aggregate comparison of the ratio of imputation accuracy between SEM_{10} and SEM_1.

Figure 10: Experimental comparison between SEM_1, SEM_5 snd SEM_{10}.

parent set exploration, k-MAX, and data imputation. Recall that n, k and d represent respectively the number of variables, the bound on the treewidth and the number of data points in the data set.

As for the first step, the size of the search space of possible parent sets for each variable is $\binom{n}{k}$ (without loss of generality, the search can be restricted to parent sets with a size up to the chosen treewidth). The complete exploration of this space would be unfeasible except for toy examples. The algorithm BIC* ([2]) was designed exactly to overcome this complexity blowup by guiding the exploration to the most promising parent sets in the allowed time. In our implementation, we choose to allow at most t seconds for the exploration of each variable. The final cost of this step is then $O(n)$. It should be noted that in theory the number of parent sets explored in this allowed time depends also by d, as for a large data sets one has time to compute fewer scores. In practice however one can implement the computation of the score of a parent set with smart intersections on pre-processed arrays, greatly diminishing the

impact of d. Such was the case in our experiment, where the increase in d had a negligible effect on the size of the final set of un-pruned parent sets returned by the algorithm (only the un-pruned parent sets impact on the performance of the whole process; the sub-optimal parent sets are discarded). However, in a practical application of SEM-kMAX that comprises also massive data sets (millions of data points) one could choose to assign to the first step a maximum execution time that depends on both n and d, with a final cost of $O(nd)$.

The second step is the execution of k-MAX. This is an iterative algorithm. For it to fully explore the search space it would require $\binom{n}{k+1}$ iterations, as it would have to select all possible combinations of $k+1$ initial cliques from the n nodes. Again, the complete exploration of this space is unfeasible, and we are forced to set a maximum execution time. In our implementation we choose to allow at most $t \cdot n/10$ seconds. In every iteration, we first build a core structure of k nodes. Since k is constant by choice, the related computation takes constant time too. The subsequent part of the algorithm adds one node at a time; for each node we explore all (in the worst case) the pre-computed parent sets for that node. As we discussed before, the size of the cache of pre-computed parent sets depends on n. Therefore the overall complexity of one iteration is $O(n)$.

The last step is data imputation, which amounts to performing an exact inference on the bounded-treewidth BN for every record containing missing values. In general the number of queries will then be $O(d)$. The cost of performing each query is $O(n\bar{v}^k)$, where $\bar{v}$ is the maximum number of states for a variable in the network. In our application k is a constant by definition, and $\bar{v}$ can be regarded as a constant too, given that it usually does not exceed few tens in applications. The complexity cost of a query is then $O(n)$, and the final cost of performing data imputation is then $O(nd)$.

So far we have considered a single iteration of SEM-kMAX. The number of iterations is usually not very large and in any case can be chosen up to a certain maximum amount. Taking this into account, the overall complexity of SEM-kMAX is $O(nd)$ or, in other words, SEM-kMAX has worst-case complexity linear in the input size (the data).

7.5. Parallelization

The whole learning procedure can greatly benefit from parallelization. In the parent set identification step, each variable can be considered independently from the others. In the structure optimization k-MAX can be run simultaneously on multiple cores, taking into consideration only the best structures found for each of them. Finally in the expectation phase the queries required for the estimation of the missing values can be executed independently for each data point.

8. Conclusions

We presented a new anytime algorithm (k-MAX) for learning bounded-treewidth Bayesian networks. Experiments on complete data sets show that

k-MAX finds structures with significantly higher fit to the data than its competitors, especially on high-dimensional data sets. Moreover, k-MAX can be plugged within structural EM in order to perform structure learning from incomplete data sets; in this case it allows to efficiently compute the expectation phase thanks to the bounded treewidth. Structural EM with k-MAX achieves comparable accuracy to state-of-the-art imputation approaches based on random forest, while allowing for a speedup of about one order of magnitude. To the best of our knowledge, our approach is the first implementation of structural EM able to efficiently scale to thousands of variables.

9. Acknowledgments

References

[1] A. Darwiche, Modeling and Reasoning with Bayesian Networks, 2009.

[2] M. Scanagatta, C. P. de Campos, G. Corani, M. Zaffalon, Learning Bayesian networks with thousands of variables, in: Advances in Neural Information Processing Systems 28, 2015, pp. 1855–1863.

[3] M. Scanagatta, G. Corani, C. P. de Campos, M. Zaffalon, Learning treewidth-bounded Bayesian networks with thousands of variables, in: Advances in Neural Information Processing Systems 29, 2016, pp. 1462–1470.

[4] F. Petitjean, G. I. Webb, Scaling log-linear analysis to datasets with thousands of variables, in: Proceedings of the 15th SIAM International Conference on Data Mining, 2015, pp. 469–477.

[5] G. I. Webb, F. Petitjean, A multiple test correction for streams and cascades of statistical hypothesis tests, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1225–1264.

[6] J. H. P. Kwisthout, H. L. Bodlaender, L. C. van der Gaag, The necessity of bounded treewidth for efficient inference in Bayesian networks, in: Proceedings of the 19th European Conference on Artificial Intelligence, 2010, pp. 237–242.

[7] J. Korhonen, P. Parviainen, Exact learning of bounded treewidth Bayesian networks, in: Artificial Intelligence and Statistics, 2013, pp. 370–378.

[8] P. Parviainen, H. S. Farahani, J. Lagergren, Learning bounded treewidth Bayesian networks using integer linear programming, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, 2014, pp. 751–759.

[9] J. Berg, M. Järvisalo, B. Malone, Learning optimal bounded treewidth Bayesian networks via maximum satisfiability, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, 2014, pp. 86–95.

[10] S. Nie, C. P. de Campos, Q. Ji, Learning bounded treewidth Bayesian networks via sampling, in: Proceedings of the 13th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2015, pp. 387–396.

[11] S. Nie, C. P. de Campos, Q. Ji, Learning Bayesian networks with bounded treewidth via guided search, in: Proceedings of the 30th AAAI Conference on Artificial Intelligence, 2016, pp. 3294–3300.

[12] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Machine Learning 29 (1997) 131–163.

[13] D. J. Stekhoven, P. Bühlmann, MissForest non-parametric missing value imputation for mixed-type data, Bioinformatics 28 (1) (2012) 112–118.

[14] G. Elidan, S. Gould, Learning bounded treewidth Bayesian networks, Journal of Machine Learning Research 9 (2008) 2699–2731.

[15] H. P. Patil, On the structure of k-trees, Journal of Combinatorics, Information and System Sciences (1986) 57–64.

[16] Z. Liu, B. Malone, C. Yuan, Empirical evaluation of scoring functions for Bayesian network model selection, BMC Bioinformatics 13 (15) (2012) 1–16.

[17] C. P. de Campos, Z. Zeng, Q. Ji, Structure learning of Bayesian networks using constraints, in: Proceedings of the 26th International Conference on Machine Learning, 2009, pp. 113–120.

[18] J. Cussens, Bayesian network learning with cutting planes, in: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence, 2011, pp. 153–160.

[19] M. Scanagatta, G. de Corani, M. Zaffalon, Improved local search in Bayesian networks structure learning, in: Proceedings of the 3rd Workshop on Advanced Methodologies for Bayesian Networks, 2017, pp. 45–56.

[20] C. P. de Campos, Q. Ji, Efficient structure learning of Bayesian networks using constraints, Journal of Machine Learning Research 12 (2011) 663–689.

24

[21] A. Rooshenas, D. Lowd, Learning sum-product networks with direct and indirect variable interactions, in: Proceedings of the 31st International Conference on Machine Learning, 2014, pp. 710–718.

[22] A. E. Raftery, Bayesian model selection in social research, Sociological Methodology 25 (1995) 111–164.

[23] R. Mateescu, K. Kask, V. Gogate, R. Dechter, Join-graph propagation algorithms, Journal of Artificial Intelligence Research 37 (1) (2010) 279–328.

[24] F. Petitjean, G. I. Webb, A. E. Nicholson, Scaling log-linear analysis to high-dimensional data, in: Proceedings of the 13th IEEE International Conference on Data Mining, 2013, pp. 597–606.

[25] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, 2009.

[26] N. Friedman, M. Goldszmidt, Learning Bayesian networks with local structure, in: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence, 1996, pp. 252–262.

[27] N. Friedman, The Bayesian structural EM algorithm, in: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 1998, pp. 129–138.

[28] R. Samdani, M.-W. Chang, D. Roth, Unified expectation maximization, in: Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2012, pp. 688–698.

[29] W. Cheng, E. Hüllermeier, K. J. Dembczynski, Bayes optimal multilabel classification via probabilistic classifier chains, in: Proceedings of the 27th International Conference on Machine Learning, 2010, pp. 279–286.

[30] A. Antonucci, G. Corani, D. Mauá, S. Gabaglio, An ensemble of Bayesian networks for multilabel classification, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, 2013, pp. 1220–1225.

[31] M. Di Zio, M. Scanu, L. Coppola, O. Luzi, A. Ponti, Bayesian networks for imputation, Journal of the Royal Statistical Society: Series A (Statistics in Society) 167 (2) (2004) 309–322.

[32] F. Tang, H. Ishwaran, Random forest missing data algorithms, Statistical Analysis and Data Mining 10 (6) (2017) 363–377.